Algorithms for solving parity games

Krishna Deepak

Indian Institute of Technology, Mumbai

krishnadeepak@iitb.ac.in

November 27, 2014

Table of Contents

Definition

A parity game $G = (V0, V1, E, p)$ is composed of two disjoint sets of vertices $V0$ and $V1$, a set of directed edges $E \subseteq V \times V$, where $V = V0 \cap V1$, and a priority function $p : V0 \cup V1 \to N$, defined on its vertices. Every vertex $u \in V$ has at least one outgoing edge $(u, v) \in E$. The game is played by two players: Even, also referred to as Player 0, and Odd, also referred to as Player 1

Parity Games(Continued)

The game starts at some vertex $v_0 \in$ V . The players construct an infinite path (a play) as follows.
Let u be the last vertex added so far to the path. If u $\in$ V0, then Player 0 chooses an edge (u, v) $\in$ E. Otherwise, if u $\in$ V1, then Player 1 chooses an edge (u, v) $\in$ E. So, vertex v is added to the path, and a new edge is then chosen by either Player 0 or Player 1. As each vertex has at least one outgoing edge, the path constructed can always be continued.
Let v_0, v_1, v_2, ... be the infinite path constructed by the two players, and let p(v_0), p(v_1), p(v_2), ... be the sequence of the priorities of the vertices on the path. Player 0 wins the play if the largest priority seen infinitely many times is even, and Player 1 wins otherwise.

Definitions

- A *strategy* for Player i in a game G specifies, for every finite path v_0, v_1, ..., v_k in G that ends in a vertex $v_k \in$ Vi, an edge $(v_k, v_{k+1}) \in$ E
- The strategy is said to be a *positional strategy* if the edge (v_k, v_{k+1}) $\in$ E chosen depends only on v_k, the last vertex visited
- A strategy for Player i is said to be a *winning strategy* if using this strategy ensures a win for Player i, no matter which strategy is used by the other player
- The *winning set* for Player i, denoted by $win_i(G)$, is the set of vertices of the game from which Player i has a winning strategy. By the Determinacy Theorem for parity games we have that $win_0(G) \cup win_1(G) =$ V

Overview of the new Algorithm

The previously known deterministic algorithm(Zielonka's algorithm) on which our improvement is built will be described fully later.
It has a recursive structure: solving a game with n vertices may require two recursive calls to smaller games. In the worst case, each of these games may have $n - 1$ vertices, resulting in a running time satisfying the recurrence T(n) = 2T($n - 1$)+O(n^2), which yields T(n) = O(2^n). We offer no improvement in the first of the two recursive calls but we do take advantage of a special feature of the second of these which leads to running time T(n) = T($n - 1$) + T($n - l$) + O(n^l) for some l = l(n).
With an appropriate choice of l(n), we achieve our subexponential algorithm with running time $n^{O(\sqrt{n})}$.
We would first describe the Zielonka's Algorithm and later extend it.

Preliminaries

A set B $\in$ V is said to be *i-closed*, where i $\in \{0, 1\}$, if for every u $\in$ B:

- if u $\in$ Vi then there is some (u, v) $\in$ E, such that v $\in$ B; and
- if u $\in$ V$\neg$i then for every (u, v) $\in$ E, we have v $\in$ B.

(We use $\neg$i for the element $(1 - i)$ in $\{0, 1\}$) In other words, a set B is i-closed if Player i can always choose to stay in B while Player $\neg$i cannot escape from it, i.e., B is a trap for Player $\neg$i.

Let A $\in$ V be an arbitrary set. The *i-reachability* set of A, denoted $reach_i$(A), contains all vertices in A and all vertices from which Player i has a strategy to enter the set A at least once; we call such a strategy an i-reachability strategy to set A.

Some Lemmas

Lemma 1

For each $i \in \{0, 1\}$, the set $win_i(G)$ is i-closed.(Proof - Straightforward)

Lemma 2

For every set $A \in V$ and $i \in \{0, 1\}$, the set $V \setminus reach_i(A)$ is $(\neg i)$-closed.

Proof.

Let $u \in V \setminus reach_i(A)$. Recall that every vertex has at least one outgoing edge, hence if $u \in V\neg i$ then there must be an edge $(u, v) \in E$ from vertex u into the set $V \setminus reach_i(A)$, i.e., such that $v \notin reach_i(A)$, since otherwise vertex u would be in $reach_i(A)$. Similarly, if $u \in Vi$ then all edges from vertex u must go into the set $V \setminus reach_i(A)$. Therefore, the set $V \setminus reach_i(A)$ is $(\neg i)$-closed. □

Sub Game

If B $\in$ V is such that for every vertex u $\in$ V $\setminus$B there is an edge (u, v) with v $\in$ V $\setminus$B, then the *subgame* G $\setminus$B is the game obtained from G by removing the vertices of B and all the edges that touch them. We will only be using B's for which V $\setminus$B is an i-closed set, for some i.

Lemma 3

Let G' be a subgame of G and let i $\in \{0, 1\}$. If V', the vertex set of G', is i-closed in G, then win_i(G') $\subseteq$ win_i(G).

Proof.

A winning strategy for Player i from the set win_i(G') in the subgame G' is also winning for her from the same set in the original game G. Player $\neg$i cannot escape to V $\setminus$V', since the set V' is i-closed in G. $\square$

Lemma 4

Let G be a parity game, let $i \in \{0, 1\}$ and $j = \neg i$. If $U \subseteq win_j(G)$ and $U' = reach_j(U)$, then $win_j(G) = U' \cup win_j(G \setminus U')$ and $win_i(G) = win_i(G \setminus U')$.

Proof.

Let W_j = U' $\cup$ win_j(G $\setminus$U') and W_i = win_i(G $\setminus$U'); Since (W_i,W_j) is a partition of V , it suffices to show that $W_i \subseteq win_i$(G) and $W_j \subseteq win_j$(G). By Lemma 2, V $\setminus$U, the vertex set of G $\setminus$U, is i-closed. The first inclusion then follows from Lemma 3.

Proof (Cont.)

To show the second inclusion, we exhibit a strategy for Player j that is winning for her from the set W_j in the game G. By the assumption that U $\subseteq$ win_j(G), there is a strategy σ for Player j in the game G which is winning for her from all vertices in U. Let τ be a winning strategy for Player j from the set win_j(G \U') in the subgame G \U'. A strategy π for Player j in the game G is made by composing strategies σ and τ in the following way: if the play so far is contained in the set win_j(G \U') then follow strategy τ , otherwise use the j-reachability strategy to the set U and restart the play following the strategy σ thenceforth. The strategy π is well-defined because, by Lemma 1, Player i can escape from winj(G \U') only into the set U'. By prefix independence of parity games, the strategy is a winning strategy for Player j, because if it ever switches from following into following then an infinite suffix of the play is winning for Player j. □

Lemma 5

Let G be a parity game. Let $d = d(G)$ be the highest priority and let $A = A_d(G)$ be the set of vertices of highest priority. Let $i = d \bmod 2$ and $j = \neg i$. Let $G' = G \setminus reach_i(A)$. Then, we have $win_j(G') \subseteq win_j(G)$. Also, if $win_j(G') = \phi$; then $win_i(G) = V(G)$, i.e., Player i wins from every vertex of G.

Proof.

That win_j(G') $\subseteq$ win_j(G) follows from Lemmas 2 and 3. Suppose now that win_j(G') $= \phi$. Let τ be a winning strategy for Player i from win_i(G') (which, by determinacy, is equal to V $\setminus reach_i$(A)) in the subgame G'. We construct a strategy π for Player i in the following way: if a play so far is contained in the set win_i(G') then follow strategy τ ; otherwise the current vertex is in $reach_i$(A) so follow the i-reachability strategy to the set A; moreover, each time the play re-enters the set win_i(G') restart the play and follow strategy τ. If a play following the strategy π visits $reach_i$(A) (and hence A) infinitely often then it is winning for Player i because i = d mod 2. Otherwise, it has an infinite suffix played according to strategy τ, and hence it is winning for Player i by prefix independence of parity games. □

Zielonka's Exponential Algorithm

Algorithm win(G)
if V(G) = ϕ; **then return** (ϕ, ϕ)
d = d(G) ; A = A_d(G)
i = d mod 2 ; j = ¬i
(W_0',W_1') = win(G \ $reach_i$(A))
if W_j' = ϕ
    then (W_i, W_j) = (V(G), ϕ)
else
    (W_0",W_1") = win(G \ $reach_j$(W_j'))
    (W_i, W_j) = (W_i",V(G) \ W_i")
endif
return (W_0, W_1)

Proof

Algorithm win(G) is based on Lemmas 4 and 5. It starts by letting d be the largest priority in G and by letting A be the set of vertices having this highest priority. Let i = d mod 2 be the index of the player associated with the highest priority, and let j = ¬i be the index of the other player. The algorithm first finds the winning sets (W_0',W_1') of the smaller game G' = G \reachi(A), using a recursive call; By Lemma 5, if W_j' $= \phi$; then Player i wins from all vertices of G and we are done. Otherwise, again by Lemma 5, we know that $W_j' \subseteq win_j$(G). The algorithm then finds the winning sets (W_0",W_1") of the smaller game G" = G $\setminus reach_j(W_j')$ by a second recursive call. By Lemma 4, we then know that win_i(G) = W_i" and win_j(G) = $reach_j(W_j') \cup W_j$" = V(G) $\setminus W_i$".

Complexity

Let T(n) be the maximum running time of algorithm win(G) for a game on at most n vertices. Algorithm win(G) makes two recursive calls win(G') and win(G") on games with at most n-1 vertices. Other than that, it performs only O(n^2) operations. (The most time-consuming operations are the computations of the sets $reach_i$(A) and $reach_j(W_j')$.) Thus T(n) $\leq$ 2T(n - 1) + O(n^2). It is easy to see then that T(n) = O(2^n).

Dominions

- A set $D \subseteq V(G)$ is said to be an *i-dominion* if Player i can win from every vertex of D without ever leaving D. Note, in particular, that an i-dominion must be i-closed.

Lemma 6

Let G be a parity game on n vertices and let $\ell \leq n/3$. There is an $O(n^\ell)$-time algorithm that finds a non-empty dominion in G of size at most ℓ, or determines that no such dominion exists.

Proof.

If $\ell \leq n/3$ then, for all $j \leq \ell$, we have that $\binom{n}{j}/\binom{n}{j-1} > 2$. The number $\sum_{j=1}^{\ell} \binom{n}{j}$ of subsets of V of size at most ℓ is therefore at most $2\binom{n}{\ell}$. For each such subset U we check, whether it is 0-closed or 1-closed. If both tests fail, then U is clearly not a dominion.

Proof (Cont.)

If U is i-closed, for some i $\in \{0, 1\}$, we form the game G[U] which is the game G restricted to U. This is well-defined since U is i-closed. We now apply the exponential algorithm of the previous section to G[U] and find out, in $O(2^\ell)$ time, whether Player i can win from all the vertices of G[U]. If so, then U is an i-dominion, otherwise it is not. The total running time of the algorithm is therefore $O(\binom{n}{\ell} 2^\ell) = O(n^\ell)$, as required. $\square$

New Sub-Exponential Algorithm

We denote algorithm in lemma 6 by **dominion**(G, ℓ), and suppose that it returns either the pair (D, i) if successful, or (ϕ,-1) if not.

Algorithm new-win(G)
if V(G) = ϕ; **then return** (ϕ, ϕ)
n = V(G); $\ell = \sqrt{2n}$
(D,i) = **dominion**(G,ℓ); j = ¬i
if D = ϕ
    then (W_i, W_j) = **old-win**(G)
else
    (W_0',W_1') = **new-win**(G $\backslash reach_i$(D))
    (W_j, W_i) = (W_j',V(G) $\backslash W_j$')
endif
return (W_0, W_1)

Algorithm old-win(G)
d = d(G) ; A = A_d(G)
i = d mod 2 ; j = ¬i
(W_0',W_1') = **new-win**(G $\backslash reach_i$(A))
if W_j' = ϕ
    then (W_i, W_j) = (V(G), ϕ)
else
    (W_0",W_1") = **new-win**(G $\backslash reach_j$(W_j'))
    (W_i, W_j) = (W_i",V(G) $\backslash W_i$")
endif
return (W_0, W_1)

Complexity Analysis

If T(n) is such that for every $n > 3$, $T(n) \leq T(n-1) + T(n-\ell) + O(n^{\ell})$ where $\ell = \sqrt{2n}$, then $T(n) = n^{O(\sqrt{n})}$

Mean Payoff Games

Definition

A mean payoff game (MPG for short) is played by two adversaries, players MAX and MIN, on a finite, directed, edge-weighted, leafless graph G = (V, E, w), where $V = V_{MAX} \cup V_{MIN}$, $V_{MAX} \cap V_{MIN} = \phi$, $E \subseteq V \times V$, and $w : E \to Z$ is the weight function.

Starting from some initial vertex, the players move a pebble along edges of the graph. Whenever, the pebble comes to a vertex $v \in V_{MAX}$, player MAX makes a move by selecting some edge (v, u) and the pebble goes to vertex u. Similarly, MIN selects a move when the pebble is in a vertex from V_{MIN}. The duration of the game is infinite and the resulting infinite sequence of edges e1, e2, e3 . . . is called a play. Player MAX wants to maximize the payoff $v_{max} = \liminf_{k\to\infty} \sum_{i=1}^{k} w(ei)$ whereas player MIN wants to minimize the payoff $v_{min} = \limsup_{k\to\infty} \sum_{i=1}^{k} w(ei)$

Algorithmic problems for MPGs

p-mean partition problem

Given p, partition the vertices of an MPG G into subsets $G_{\leq p}$ and $G_{>p}$ such that MAX can secure a payoff $>$ p starting from every vertex in $G_{>p}$, and MIN can secure a payoff $\leq$ p starting from every vertex in $G_{\leq p}$.

The algorithm solves the 0-mean partition problem, which subsumes the p-mean partition. Indeed, subtracting p from the weight of every edge makes the mean value of all cycles (in particular, of optimal cycles) smaller by p, and the problem reduces to 0-mean partitioning.

The longest shortest paths(LSP) problem

Definition

Given a directed edge-weighted graph G with a unique sink t, and a distinguished set $U \subseteq V[G]$ of *controlled* vertices, with $t \notin U$. We have to find a positional strategy selecting exactly one outgoing edge from each vertex in U such that in the graph G the length of the shortest path from every vertex to sink t is as large as possible (over all positional strategies).

The length of a finite simple path to the sink in G equals the sum of edge weights on the path. In a cyclic G the distances to the sink are defined as
(1) $+\infty$ for every vertex on a positive weight cycle;
(2) $-\infty$ for every vertex on a negative weight cycle;
(3) 0 for every vertex on a 0-weight cycle.

Relating the 0-mean partition and LSP problems

- To find a 0-mean partition in an MPG G, add a retreat vertex t (belongs to MIN and will later become the sink) to the game graph with a self-cycle edge of weight 0, then add a 0-weight retreat edge from every MAX vertex to t. From now on, we assume that G has undergone this transformation
- Adding the retreat does not change the 0-mean partition of the game, except that the new retreat vertex belongs to the $G_{\leq 0}$ part
- Now, break the self-cycle in t and consider the LSP problem for the resulting graph, with t being the unique sink. The set V_{MAX} becomes the set of controlled vertices, and the initial strategy selects retreat t in every controlled vertex, guaranteeing that no vertex has distance $-\infty$
- The partition $G_{>0}$ consists exactly of those vertices for which the longest shortest path distance to t is $+\infty$

Algorithm for the LSP problem

- The main step of our iterative strategy improvement algorithm is the so-called **attractive switch**. Comparing a current choice made by the strategy with alternative choices, a possible improvement can be decided locally as follows. If changing the choice in a controlled vertex to another successor seems to give a longer distance (seems attractive), we make this change. Such a change is called a switch.
- Switching is done in the following way. Suppose the current distance (using the current strategy σ) from a vertex v to the sink is $d_\sigma(v)$, but for an edge (v, u) not used by σ, we have $d_\sigma(v) < w(v,u) + d_\sigma(u)$ (attractiveness). Then we switch the current edge $(v, \sigma(v))$ to (v, u), get new strategy σ, and recompute the shortest distances

- Every such switch really increases the shortest distances (i.e., attractiveness is improving or profitable)
- Once none of the alternative possible choices is attractive, all possible positive weight cycles MAX can enforce are found (i.e., a stable strategy is optimal).
- The order in which attractive switches are made is crucial for the subexponential complexity bound which is obtained by randomization scheme
- A facet corresponds to a subgame in which one of the choices of player MAX is fixed in one vertex, and choices in all other vertices are unconstrained

Randomization Scheme

The algorithm for computing the LSP in an MPG/LSP instance G is as follows:
1) Start from some strategy σ that guarantees for shortest distances $> -\infty$ in all vertices
2) If σ is the only possible MAX strategy in G, return it as optimal.
3) Otherwise, randomly and uniformly select some facet F of G not containing σ. Temporarily throw this facet away, and recursively find a best strategy σ' on the remainder, G$\setminus$F. This corresponds to deleting a MAX edge not used by σ and finding a best strategy in the resulting subgame.
4) If σ' is optimal in G, return it as a result. Optimality is easily checked by computing shortest distances in $G_{\sigma'}$ from all vertices of the graph to the sink, and testing whether there is an attractive switch from σ' to the edge defining F.
5) Otherwise, make an attractive switch to F, set G $=$ F, denote the resulting strategy by σ, and repeat from step 2.

Converting a Parity Games into Mean Payoff Game

- In a Parity Game, Player EVEN (MAX) wants to ensure that in every infinite play the largest color appearing infinitely often is even, and player ODD (MIN) tries to make it odd
- Transform a parity game into a MPG by leaving the graph and the vertex partition between players unchanged, and by assigning every vertex of parity c the weight $(-n)^c$, where n is the total number of vertices in the game.
- Apply the algorithm described in the preceding sections to find a 0-mean partition. Obviously, the vertices in the partition with $value > 0$ are winning for EVEN and all other are winning for ODD in the parity game.

The End